**DSA Lab 3:**

**Submitted by: Khuram Iqbal**

**Submitted to: Nazeef ul Haq**

**Reg #: 2022-CS-48**

**Problem 1: Collaboration With Omar Baig (2022-CS-43)**

**Part (vi):** **(a):** The logic is sound. Karatsuba’s Algorithm can function in any base by recursively multiplying components.

**(b):** The logic is correct. As the base-b representation of a number x increases, its length decreases.

**(c):** The logic is flawed. Karatsuba’s algorithm relies on three recursive calls. Even if addition and subtraction were constant-time operations (which they are not), the three recursive calls would still be the dominant factor in most cases, resulting in a total time complexity of O(n^1.58).

**Problem 2: Collaboration With Omar Baig (2022-CS-43)**

**Part (I):** **Pseudocode:**

Function friendSlower(Input):

Let temp [Input.length] be the new array

k = 0

for i: 1 to n:

for j: i+1 to n:

if Input[i][0] in Input[j]:

temp[k] = (i , j )

k = k + 1

end for loop2

end for loop1

**Description:** This algorithm employs brute force. It selects an element from the array and compares it to all subsequent elements. It does this by generating a list of numbers from the next element and checking if the first or last number from the previous element is present in that list.

**Runtime Analysis:** Since two nested loops are used for comparison, the worst-case runtime is O(n^2).

**Part (II):** **Pseudocode:**

Function friendsFaster(Input):

Let temp[Input.length] be the new array

Sorted = MergeSort(Input, 0, Input.length)

for i: 1 to n, i = i + 2:

if Input[i][0] in Input[i + 1]:

temp[k] = (i , i + 1 )

k = k + 1

**Description:** This algorithm is a modified version of the previous part. Instead of comparing with all possible combinations, it only checks with the immediate successor and increases the loop counter by 2 for each iteration.

**Runtime Analysis:** The for loop, with an increment counter of two, has a runtime of O(log n). The Merge Sort algorithm takes O(n log n) time. Therefore, the total worst-case time complexity of the algorithm is O(n log n).

**Problem 3: Collaboration With Omar Baig (2022-CS-43)**

**Part (a):** In a nested loop, a brute-force approach is used for toad-to-toad comparisons. Two toads are compared, and the result is checked against the truth value of the first toad. If it's true, the index of the first toad is appended to the list and returned.

**Part (b):** Using half the comparisons in a single loop instead of a nested loop, the toads are checked in a manner of n/2. The method of comparison remains the same, except that only the next immediate toad is checked (which is sufficient to determine trustworthiness). If the total number of trustworthy toads m is less than m/2 for the range 0 < m ≤ n/2, the list returned is empty; otherwise, the list is returned with indexes satisfying the constraints.

**Part (c):** Using half the comparisons in a single loop instead of a nested loop, the toads are checked in a dn/2e manner. The method of comparison remains the same, except that only the next immediate toad is checked (which is sufficient to determine trustworthiness). If the total number of trustworthy toads m is less than m/2 for the range 0 < m ≤ dn/2e, the list returned is empty; otherwise, the list is returned with indexes satisfying the constraints. dn/2e represents the differential for odd values.

**Part (d):** To find a single trustworthy toad, a searching algorithm is used with comparisons superimposed. With a list of indices, a divide-and-conquer approach is employed to retrieve the relevant indices one by one. In the recursive algorithm, this takes the form of a base case with an array length of 1 returning the value; otherwise, the array is split, and the function is called again with the next value.

**Part (e):** **Inductive Hypothesis:** The recursive algorithm breaks down the array, skipping the first element. The search continues in the updated array.

**Base Case:** If the array length is 0, return array [0].

**Inductive Step (Recursive Step):** Index = 1 + search (k + 1, n).

**Conclusion:** The index is found.

**Part (f):** Since all values in the array are checked, in the worst case, every value will be checked. So the algorithm has a time complexity of O(n).

**Part (g):** A loop iterates over all toad values to find trustworthy toads.Top of Form